Circuitos Lógicos

RELATÓRIO III

Discente: Rebeca de Macêdo Ferreira. 2016000524

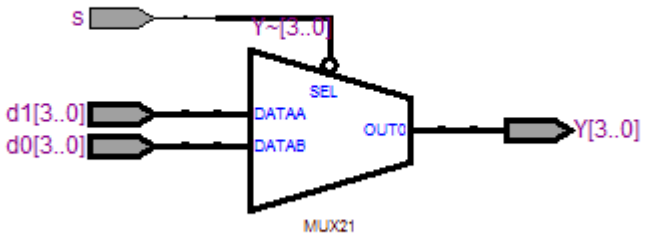
MUX 2 ENTRADAS:

Levamos pra aula um arquivo .vhd com o código referente a um mux de duas portas encontrado no livro Harris e Harris

```
1 library IEEE; use IEEE.STD_LOGIC_1164.all;
2
3 entity mux2 is
4         port(d0, d1: in STD_LOGIC_VECTOR(3 downto 0);
5                   s: in STD_LOGIC;
6                   y: out STD_LOGIC_VECTOR(3 downto 0));
7 end;
8
9 architecture synth of mux2 is
10 begin
11 y <= d0 when s='0' else d1;
12 end;
```

onde d0 e d1 são entradas (4 bits), s entrada de decisão e y saída. Quando ‘s’ = 0, y = d1; quando s = 1, y = d0.

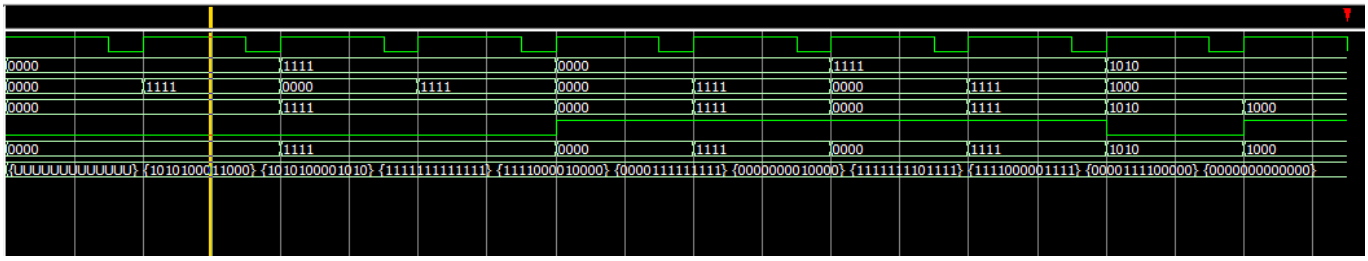
Compilamos o código no quartus e selecionamos Toos → Netlist Viewers → RTL Viewer para obter o circuito



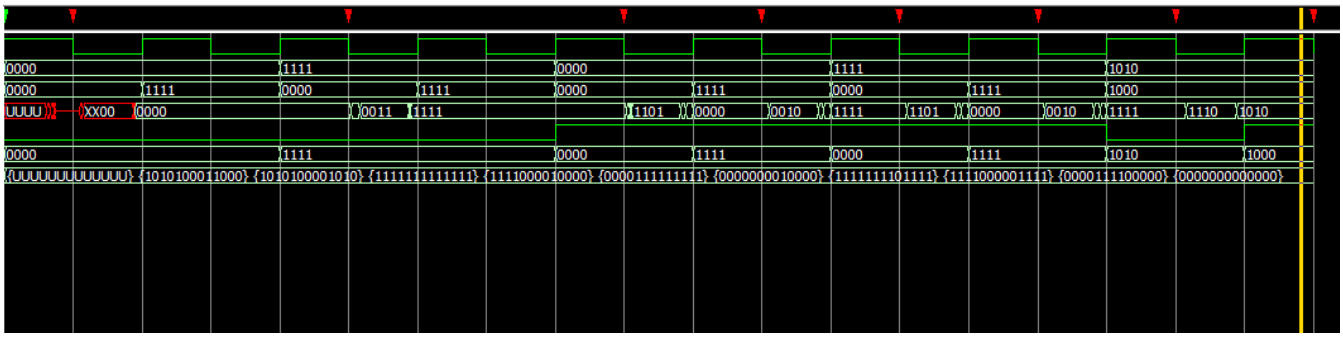
Fizemos a tabela verdade num arquivo .tv do nosso mux:

```
000000000_0000
000011110_0000
111100000_1111
111111110_1111
000000001_0000
000011111_1111
111100001_0000
111111111_1111
101010000_1010
101010001_1000
```

E adaptamos o testbench para nosso mux e simulamos o RTL level em Tools → Run EDA Simulation Tool → RTL Level, tivemos como resultado o seguinte:



Com isso fomos simular o Gate level, que considera os atrasos das portas do circuito, em Tools → Run EDA Simulation Tool → Gate Level, obtivemos o seguinte:

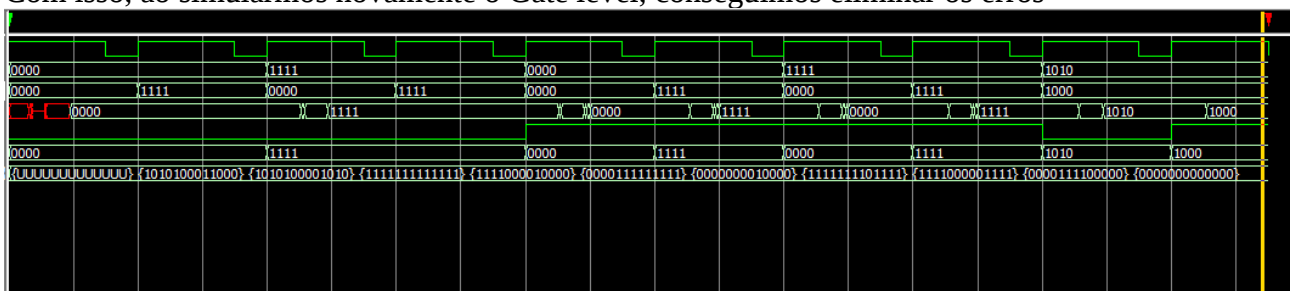


Percebemos os erros, e análogo aos exemplos anteriores, alteramos no testbench os atrasos do clock em 1, de 5 ns para 15 ns, afim de corrigir os erros.

```
27  process begin
28      clk <= '1'; wait for 5 ns;
29      clk <= '0'; wait for 5 ns;
30  end process;
```

```
27  process begin
28      clk <= '1'; wait for 15 ns;
29      clk <= '0'; wait for 5 ns;
30  end process;
```

Com isso, ao simularmos novamente o Gate level, conseguimos eliminar os erros



Visualização em lista do mux de 2 entradas:

ps	delta	/testbench_mux/s	/testbench_mux/clk	/testbench_mux/d0	/testbench_mux/d1	/testbench_mux/yexpected	/testbench_mux/y
0	+0	U	U	UUUU	UUUU	UUUU	UUUU
0	+1	U	1	UUUU	UUUU	UUUU	UUUU
0	+2	0	1	0000	1111	UUUU	0000
0	+3	0	1	0000	1111	0000	0000
5000	+1	0	0	0000	1111	0000	0000
10000	+1	0	1	0000	1111	0000	0000
10000	+2	0	1	1111	0000	0000	1111
10000	+3	0	1	1111	0000	1111	1111
15000	+1	0	0	1111	0000	1111	1111
20000	+1	0	1	1111	0000	1111	1111
20000	+2	1	1	1111	0000	1111	0000
20000	+3	1	1	1111	0000	0000	0000
25000	+1	1	0	1111	0000	0000	0000
30000	+1	1	1	1111	0000	0000	0000
30000	+2	1	1	0000	1111	0000	1111
30000	+3	1	1	0000	1111	1111	1111
35000	+1	1	0	0000	1111	1111	1111

MUX DE 4 ENTRADAS

Vamos agora fazer a descrição do multiplex de 4 entradas de forma análoga ao anterior. O código em vhd utilizado é do livro Harris e Harris.

mux4.vhd

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.all;
3
4  entity mux4 is
5  port(d0, d1, d2, d3: in STD_LOGIC_VECTOR(3 downto 0);
6       s: in STD_LOGIC_VECTOR(1 downto 0);
7       y: out STD_LOGIC_VECTOR(3 downto 0));
8  end;
9
10 architecture synth1 of mux4 is
11 begin
12 y <= d0 when s = "00" else
13      d1 when s = "01" else
14      d2 when s = "10" else
15      d3;
16 end;
```

Nesse mux as entradas são d0, d1, d2, d3 (todos 4 bits), a entrada decisão é o s (2 bits), y é a saída (4 bits).
Compilamos o código no quartus e selecionamos Toos → Netlist Viewers → RTL Viewer para obter o circuito:

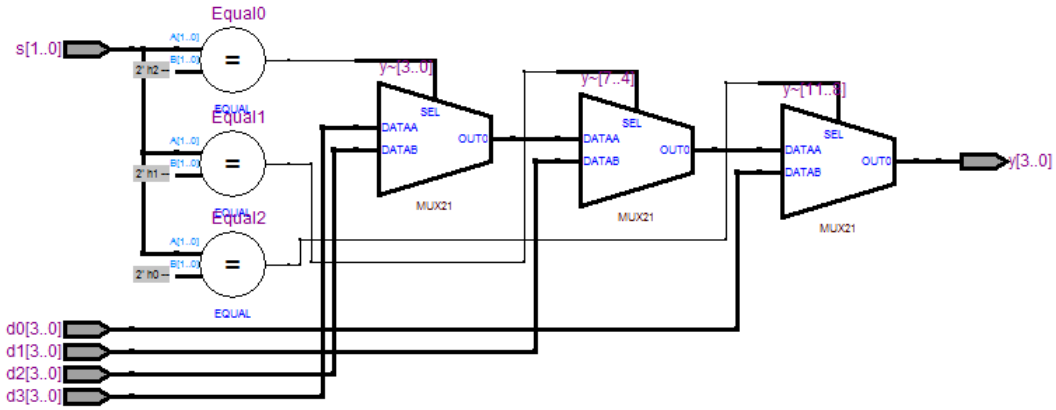


Tabela verdade feita pro mux de 4 entradas

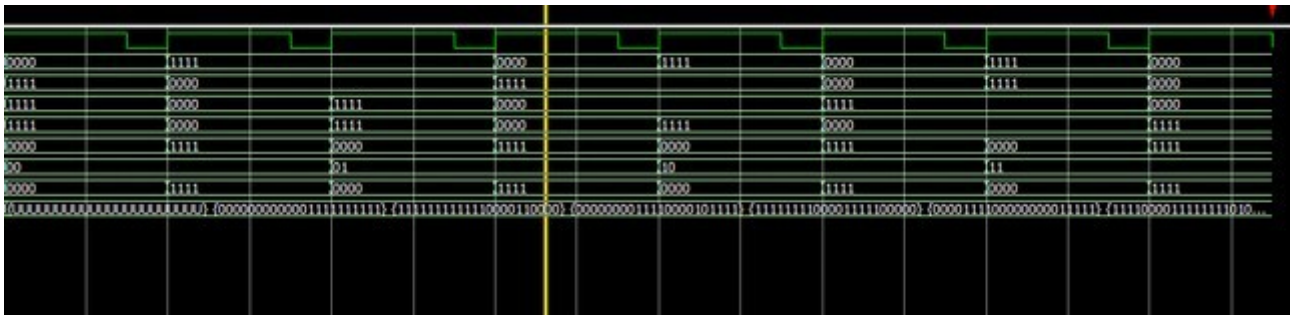
```
0000111111111111100_0000
1111000000000000000_1111
1111000011111111101_0000
0000111100000000001_1111
1111111100001111110_0000
0000000011110000010_1111
1111111111111000011_0000
0000000000000111111_1111
```

Pegamos um testbench já feita em aulas anteriores e adaptamos pro nosso mux, e como sempre adicionamos ele no Quartus para fazermos as simulações

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_arith.ALL;
4  use IEEE.STD_LOGIC_unsigned.ALL;
5  use STD.TEXTIO.ALL ;
6
7  entity testbench_mux4 is -- no inputs or outputs
8  end;
9  architecture sim of testbench_mux4 is
10     component mux4
11         port (d0, d1, d2, d3: in STD_LOGIC_VECTOR(3 downto 0);
12                 s: in STD_LOGIC_VECTOR(1 downto 0);
13                 y: out STD_LOGIC_VECTOR(3 downto 0));
14     end component;
15
16 signal clk: STD_LOGIC;
17 signal d0, d1, d2, d3, y: STD_LOGIC_VECTOR(3 downto 0);
18 signal s:  STD_LOGIC_VECTOR(1 downto 0);
19 signal yexpected: STD_LOGIC_VECTOR(3 downto 0);
20 constant MEMSIZE: integer := 8;
21 type tvarray is array (MEMSIZE downto 0) of STD_LOGIC_VECTOR (21 downto 0);
22 signal testvectors: tvarray;
23 shared variable vectornum, errors: integer;
24 begin
25 -- instantiate device under test
26 dut: mux4 port map (d0, d1, d2, d3, s, y);
27 -- generate clock
```

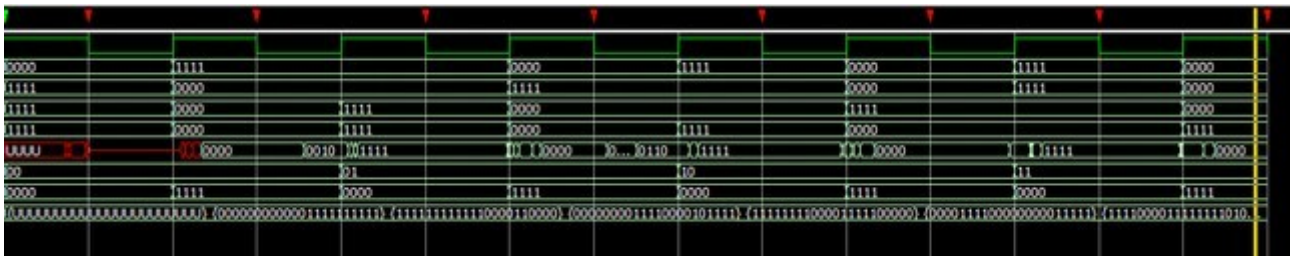
Apenas um pedaço do testbench

Simulamos então o RTL level em Tools → Run EDA Simulation Tool → RTL Level, tivemos um resultado sem nenhum erro:



```
# ** Failure: Just kidding --8tests completed successfully.
#    Time: 155 ns  Iteration: 1  Process: /testbench_mux4/line__72 File: D:/CIRCUITOS/mux4/testbench/testbench_mux4.vhd
# Break in Process line__72 at D:/CIRCUITOS/mux4/testbench/testbench_mux4.vhd line 88
```

Então fomos simular o Gate level, já esperamos que tenha alguns erros por causa de atrasos nas portas…
Tools → Run EDA Simulation Tool → Gate Level



```
Transcript
#    Time: 75 ns  Iteration: 1  Instance: /testbench_mux4
# ** Error: Vetor deu erro n. Teste: 7. Esperado yesp ='1'Valor Obtido: y(3) ='0'
#    Time: 75 ns  Iteration: 1  Instance: /testbench_mux4
# ** Failure: 8tests completed, errors = 32
#    Time: 75 ns  Iteration: 1  Process: /testbench_mux4/line__72 File: D:/CIRCUITOS/mux4/testbench/testbench_mux4.vhd
# Break in Process line__72 at D:/CIRCUITOS/mux4/testbench/testbench_mux4.vhd line 93
```

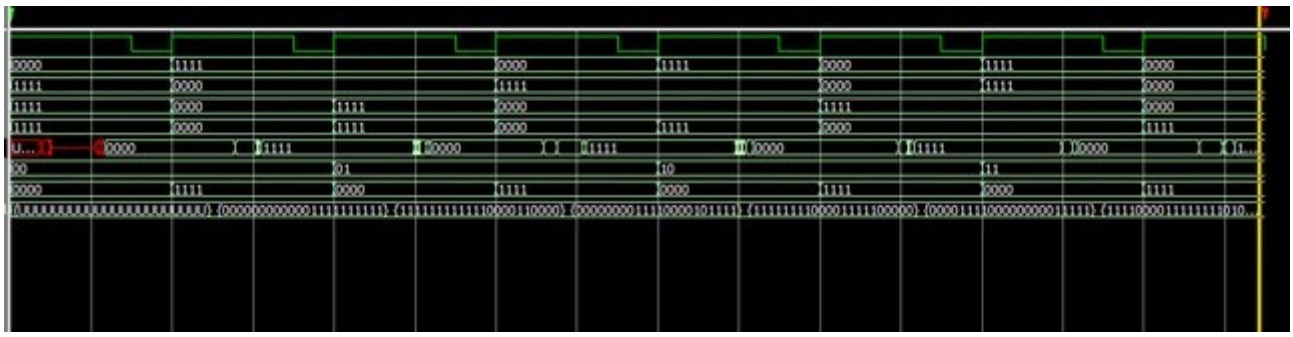
Mudamos os clocks em 1 no testbench que inicialmente estava com 5ns, depois mudamos pra 10 e depois pra 15…

```
28  process begin
29      clk <= '1'; wait for 5
30      clk <= '0'; wait for 5
31  end process;
```

```
28  process begin
29      clk <= '1'; wait for 10 ns;
30      clk <= '0'; wait for 5 ns;
31  end process;
```

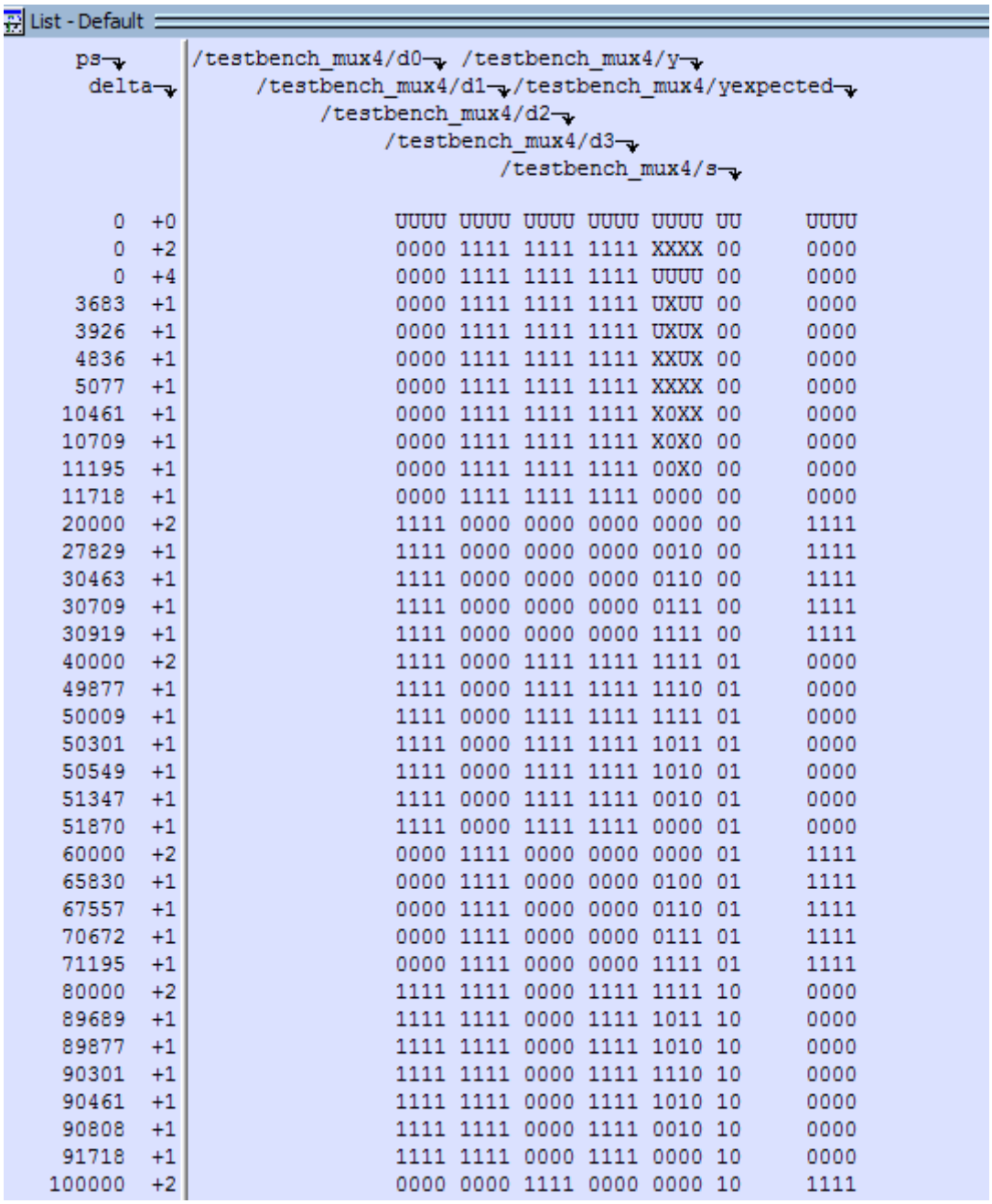
```
28  process begin
29      clk <= '1'; wait for 15 ns;
30      clk <= '0'; wait for 5 ns;
31  end process;
```

Só com o clock em 1 com 15ns o Gate level ficou sem erros na simulação



```
Transcript
# view signals
# .main_pane.objects.interior.cs.body.tree
# run -all
# ** Failure: Just kidding --8tests completed successfully.
#    Time: 155 ns  Iteration: 1  Process: /testbench_mux4/line__72 File: D:/CIRCUITOS/mux4/testbench/testbench_mux4.vhd
# Break in Process line__72 at D:/CIRCUITOS/mux4/testbench/testbench_mux4.vhd line 88
```

Visualização em lista do mux de 4 entradas:



List - Default

ps	delta	/testbench_mux4/d0	/testbench_mux4/d1	/testbench_mux4/d2	/testbench_mux4/d3	/testbench_mux4/s	/testbench_mux4/y	/testbench_mux4/yexpected
0	+0	UUUU	UUUU	UUUU	UUUU	UU	UUUU	
0	+2	0000	1111	1111	1111	XXXX	00	0000
0	+4	0000	1111	1111	1111	UUUU	00	0000
3683	+1	0000	1111	1111	1111	UXUU	00	0000
3926	+1	0000	1111	1111	1111	UXUX	00	0000
4836	+1	0000	1111	1111	1111	XXUX	00	0000
5077	+1	0000	1111	1111	1111	XXXX	00	0000
10461	+1	0000	1111	1111	1111	X0XX	00	0000
10709	+1	0000	1111	1111	1111	X0X0	00	0000
11195	+1	0000	1111	1111	1111	00X0	00	0000
11718	+1	0000	1111	1111	1111	0000	00	0000
20000	+2	1111	0000	0000	0000	0000	00	1111
27829	+1	1111	0000	0000	0000	0010	00	1111
30463	+1	1111	0000	0000	0000	0110	00	1111
30709	+1	1111	0000	0000	0000	0111	00	1111
30919	+1	1111	0000	0000	0000	1111	00	1111
40000	+2	1111	0000	1111	1111	1111	01	0000
49877	+1	1111	0000	1111	1111	1110	01	0000
50009	+1	1111	0000	1111	1111	1111	01	0000
50301	+1	1111	0000	1111	1111	1011	01	0000
50549	+1	1111	0000	1111	1111	1010	01	0000
51347	+1	1111	0000	1111	1111	0010	01	0000
51870	+1	1111	0000	1111	1111	0000	01	0000
60000	+2	0000	1111	0000	0000	0000	01	1111
65830	+1	0000	1111	0000	0000	0100	01	1111
67557	+1	0000	1111	0000	0000	0110	01	1111
70672	+1	0000	1111	0000	0000	0111	01	1111
71195	+1	0000	1111	0000	0000	1111	01	1111
80000	+2	1111	1111	0000	1111	1111	10	0000
89689	+1	1111	1111	0000	1111	1011	10	0000
89877	+1	1111	1111	0000	1111	1010	10	0000
90301	+1	1111	1111	0000	1111	1110	10	0000
90461	+1	1111	1111	0000	1111	1010	10	0000
90808	+1	1111	1111	0000	1111	0010	10	0000
91718	+1	1111	1111	0000	1111	0000	10	0000
100000	+2	0000	0000	1111	0000	0000	10	1111

Figura 1: Apenas uma parte da lista do mux de 4 entradas

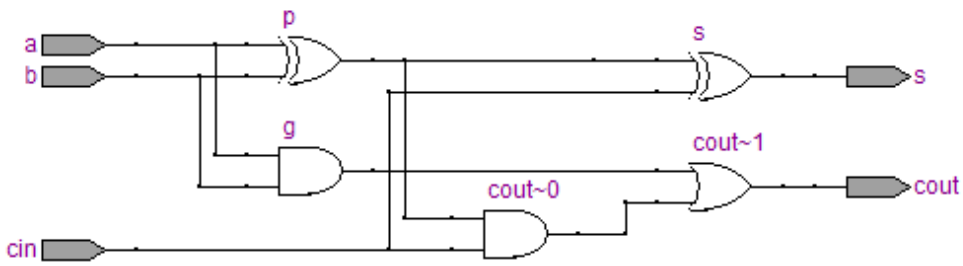
FULL ADDER

Para a descrição de um full adder, ou somador completo, usamos um codigo vdh também do livro Harris e Harris

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity fulladder is
port(a, b, cin: in STD_LOGIC;
     s, cout: out STD_LOGIC);
end;

architecture synth of fulladder is
signal p, g: STD_LOGIC;
begin
    p <= a xor b;
    g <= a and b;
    s <= p xor cin;
    cout <= g or (p and cin);
end;
```

Nesse código do full adder a, b e cin são as entrada, s é a variável que recebe a operação entre a e b, cout é o “vai um” resultante desta soma.

Selecionamos Toos → Netlist Viewers → RTL Viewer para obter o circuito de portas logicas



Fizemos a tabela verdade e salvamos como .tv :

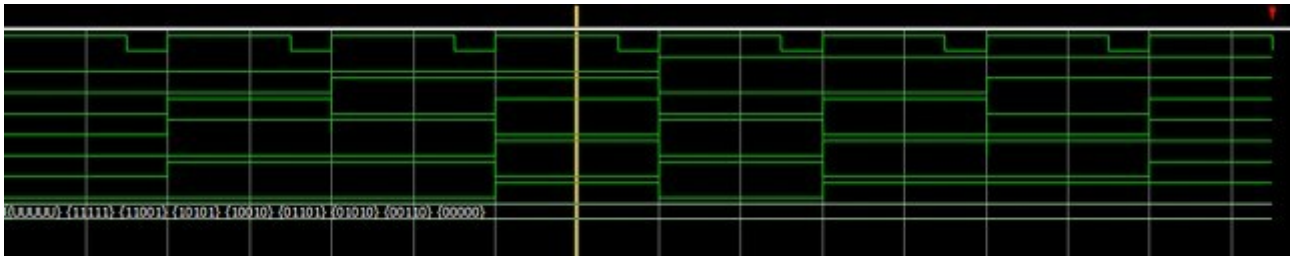
```
000_00
001_10
010_10
011_01
100_10
101_01
110_01
111_11
```

Adaptamos o testbench o mux de 4 entradas para o nosso full adder

```vhdl
-- apply test vectors on rising edge of clk
process (clk) begin
    if (clk'event and clk='1') then
        a <= testvectors (vectornum) (4);
        b <= testvectors (vectornum) (3);
        cin <= testvectors (vectornum) (2);
        sexpected <= testvectors (vectornum) (1);
        coutexpected <= testvectors (vectornum) (0);
    end if;
end process;
-- check results on falling edge of clk
process (clk) begin
    if (clk'event and clk = '0')then

        assert s = sexpected
            report "Vetor deu erro n. Teste: " &integer'image(vectornum)&". Esperado sesp ="& STD_LOGIC'image(sexpected)&"Valor Obtido: s="& STD_LOGIC'image(s);

        if (s /= sexpected) then
            errors := errors + 1;
        end if;

        assert cout = coutexpected
            report "Vetor deu erro n. Teste: " &integer'image(vectornum)&". Esperado coutesp ="& STD_LOGIC'image(coutexpected)&"Valor Obtido: cout="& STD_LOGIC'image(cout);

        if (cout /= coutexpected) then
            errors := errors + 1;
        end if;
```

Parte do testbench alterada pra o full adder

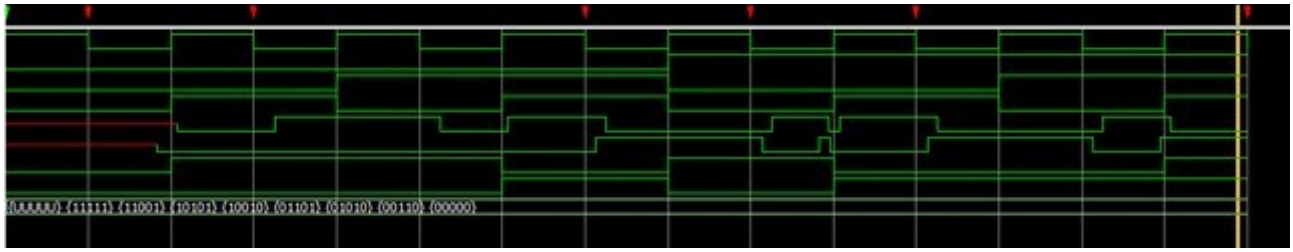
Simulamos o RTL level em RTL level em Tools → Run EDA Simulation Tool → RTL Level



```
Transcript
# view signals
# .main_pane.objects.interior.cs.body.tree
# run -all
# ** Failure: Just kidding --8tests completed successfully.
#    Time: 155 ns  Iteration: 1  Process: /testbench_fulladder/line__73 File: D:/CIRCUITOS/fulladder/testbench/testbench_fulladder.vhd
# Break in Process line__73 at D:/CIRCUITOS/fulladder/testbench/testbench_fulladder.vhd line 96
```

Para a simulação de Gate level, esperamos alguns erros devido aos clocks considerados…

Tools → Run EDA Simulation Tool → Gate Level



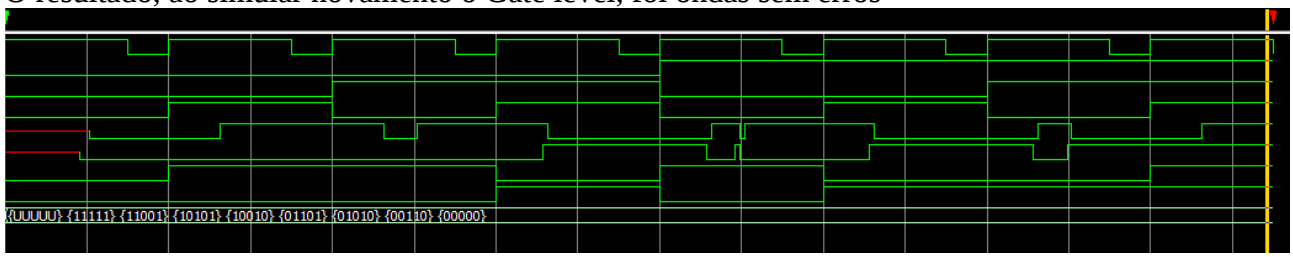
```
Transcript
#    Time: 55 ns  Iteration: 1  Instance: /testbench_fulladder
# ** Error: Vetor deu erro n. Teste: 7. Esperado sesp ='1'Valor Obtido: s='0'
#    Time: 75 ns  Iteration: 1  Instance: /testbench_fulladder
# ** Failure: 8tests completed, errors = 10
#    Time: 75 ns  Iteration: 1  Process: /testbench_fulladder/line__73 File: D:/CIRCUITOS/fulladder/testbench/testbench_fulladder.vhd
# Break in Process line__73 at D:/CIRCUITOS/fulladder/testbench/testbench_fulladder.vhd line 101
```

Melhoramos o clock em 1 no testbench, de 5ns para 15ns, visto que já tivemos esse problema em questões anteriores…

```
28  process begin
29      clk <= '1'; wait for 5 ns;
30      clk <= '0'; wait for 5 ns;
31  end process;
```

```
28  process begin
29      clk <= '1'; wait for 15 ns;
30      clk <= '0'; wait for 5 ns;
31  end process;
```

O resultado, ao simular novamento o Gate level, foi ondas sem erros



```
Transcript
# view signals
# .main_pane.objects.interior.cs.body.tree
# run -all
# ** Failure: Just kidding --8tests completed successfully.
#    Time: 155 ns  Iteration: 1  Process: /testbench_fulladder/line__73 File: D:/CIRCUITOS/fulladder/testbench/testbench_fulladder.vhd
# Break in Process line__73 at D:/CIRCUITOS/fulladder/testbench/testbench_fulladder.vhd line 96
```

Visualização em lista do full adder:

List - Default

ps / delta	/testbench_fulladder/a	/testbench_fulladder/b	/testbench_fulladder/cin	/testbench_fulladder/s	/testbench_fulladder/cout	/testbench_fulladder/sexpected	/testbench_fulladder/coutexpected
0 +0	U	U	U	U	U	U	U
0 +2	0	0	0	X	X	0	0
0 +4	0	0	0	U	U	0	0
3102 +1	0	0	0	U	X	0	0
3691 +1	0	0	0	X	X	0	0
9156 +1	0	0	0	X	0	0	0
10368 +1	0	0	0	0	0	0	0
20000 +2	0	0	1	0	0	1	0
26278 +1	0	0	1	1	0	1	0
40000 +2	0	1	0	1	0	1	0
46278 +1	0	1	0	0	0	1	0
50368 +1	0	1	0	1	0	1	0
60000 +2	0	1	1	1	0	0	1
65687 +1	0	1	1	1	1	0	1
66278 +1	0	1	1	0	1	0	1
80000 +2	1	0	0	0	1	1	0
85687 +1	1	0	0	0	0	1	0
86278 +1	1	0	0	1	0	1	0
89156 +1	1	0	0	1	1	1	0
89746 +1	1	0	0	0	1	1	0
89780 +1	1	0	0	0	0	1	0
90368 +1	1	0	0	1	0	1	0
100000 +2	1	0	1	1	0	0	1
105687 +1	1	0	1	1	1	0	1
106278 +1	1	0	1	0	1	0	1
120000 +2	1	1	0	0	1	0	1
125687 +1	1	1	0	0	0	0	1
126278 +1	1	1	0	1	0	0	1
129780 +1	1	1	0	1	1	0	1
130368 +1	1	1	0	0	1	0	1
140000 +2	1	1	1	0	1	1	1
146278 +1	1	1	1	1	1	1	1

DIRETÓRIOS UTILIZADOS

<u>Mux de 2 entradas:</u>

O arquivo com **codigo .vhd** está em: *mux2 → mux2 → mux2.vhd.*
O arquivo **testbench.vhd** está em: *mux2 → testbench → testbench_mux2.vhd.*
O arquivo com a **tabela verdade .tv** está em: *mux2 → simulation → modelsim → questao3.tv.*

<u>Mux de 4 entradas:</u>

O arquivo com **codigo .vhd** está em: mux4 → mux4 → mux4.vhd.
O arquivo **testbench.vhd** está em: mux4 → testbench → testbench_mux4.vhd.
O arquivo com a **tabela verdade .tv** está em: *mux4 → simulation → modelsim → mux4.tv.*

Full adder (somador completo):

O arquivo com **codigo .vhd** está em: fulladder → fulladder→ fulladder.vhd.
O arquivo **testbench.vhd** está em: fulladder → testbench → testbench_fulladder.vhd.
O arquivo com a **tabela verdade .tv** está em: fulladder → *simulation* → *modelsim* → *fulladder.tv*.